07 | WebSocket接口：如何测试一个完全陌生的协议接口?

2020-02-14 陈磊

接口测试实战课 进入课程 >

讲述：陈磊
时长 11:31　大小 9.24M

你好，我是陈磊。很高兴你又来和我一起探寻接口测试的奥秘了。

我们在前面一起学习了怎么分析和完成一个 HTTP 协议的接口测试，又一起学习了如何封装接口测试框架，以及如何搭建接口测试平台。我相信，现在你已经完全掌握了 HTTP 协议的接口测试了。

但是，这还不能说明你已经能独立完成接口测试了，为什么这么说呢？这是因为数据在网络传输上都是依靠一种协议来完成的，在上学的时候，你肯定学过包括 TCP、UDP、HT 等在内的一堆协议，但是如果你遇见了一个全新的协议，你知道怎么从零开始，完成接口测试吗?

今天我就以 WebSocket 为例，告诉你当你第一次接触一个完全陌生的协议接口时，你要如何开始完成接口测试工作。

未知的新协议接口并不可怕

作为一名测试工程师，在面对一个陌生协议的接口测试时，你是不是会常常感到很无助？面对这样的任务时，你的第一反应肯定是向开发工程师求助，因为开发工程师基于新协议已经完成了接口开发，向开发工程师求助显然是最好的办法。

我在工作中就遇见过类似的事情。记得那是在我参加工作的前几年，有一个被测项目的接口是一个私有协议，当我看到接口文档的时候，第一反应就是找开发工程师，向他求教一下这个私有协议。这个开发工程师人很好，他给了我一个学习脉络，其中包含了协议的说明文档、代码开发文档、实现代码等内容，我拿到这些资料后，马上按照上面他给出的学习顺序投入学习。

但是后来，在项目从交付测试到完成测试后，我发现自己走了一个弯路。因为作为测试工程师，我们不需要了解协议底层的原理，只需要了解新协议是如何传输数据，又如何返回数据库就可以了。也就是说，我们要想模拟一个客户端去验证服务端的逻辑，那么开始接口测试最快速的方法不是去看协议的说明文档，而是直接去看开发实现的客户端代码，这对于我们来说，能更直接地解决问题。但这也并不是说，那位开发工程师告诉我的学习脉络是错误的，只不过它并不是一个非常适合测试工程师的学习方法。

那在面对一个陌生的新协议时，测试工程师的首要任务是什么呢？

在我看来，就是要测试接口的正确逻辑、错误逻辑是否满足最初的需求，因此，我们需要快速地掌握验证手段。在时间紧迫的情况下，如果我们还是先学习新协议的基础知识，再学习怎么使用它，就无疑压榨了测试的工期，也会让我们在真正开始工作时手忙脚乱。

所以，我们要从解决实际问题的角度出发，直接拿到开发工程师提供的调用客户端代码，这样我们就可以快速完成工作了；在完成工作的后续时间里，我们也可以慢慢补充基础知识。这里需要你注意的是，我并不是说基础知识不重要，而是说在项目进行过程中，学习基础知识很多时候没有完成项目的质量保障工作重要。

第一次接触 WebSocket 接口

我在前面说了一大堆方法论，你看到后可能还是摸不到头脑，那么现在，我就以一个我亲身经历的例子来告诉你，面对一个陌生协议接口要怎么去做测试。

大概是在 2017 年，我第一次接触到 WebSocket 协议的接口，当开发工程师告诉我这是一个 WebSocket 的接口时，我一脸懵，完全不知道要如何开始测试它。

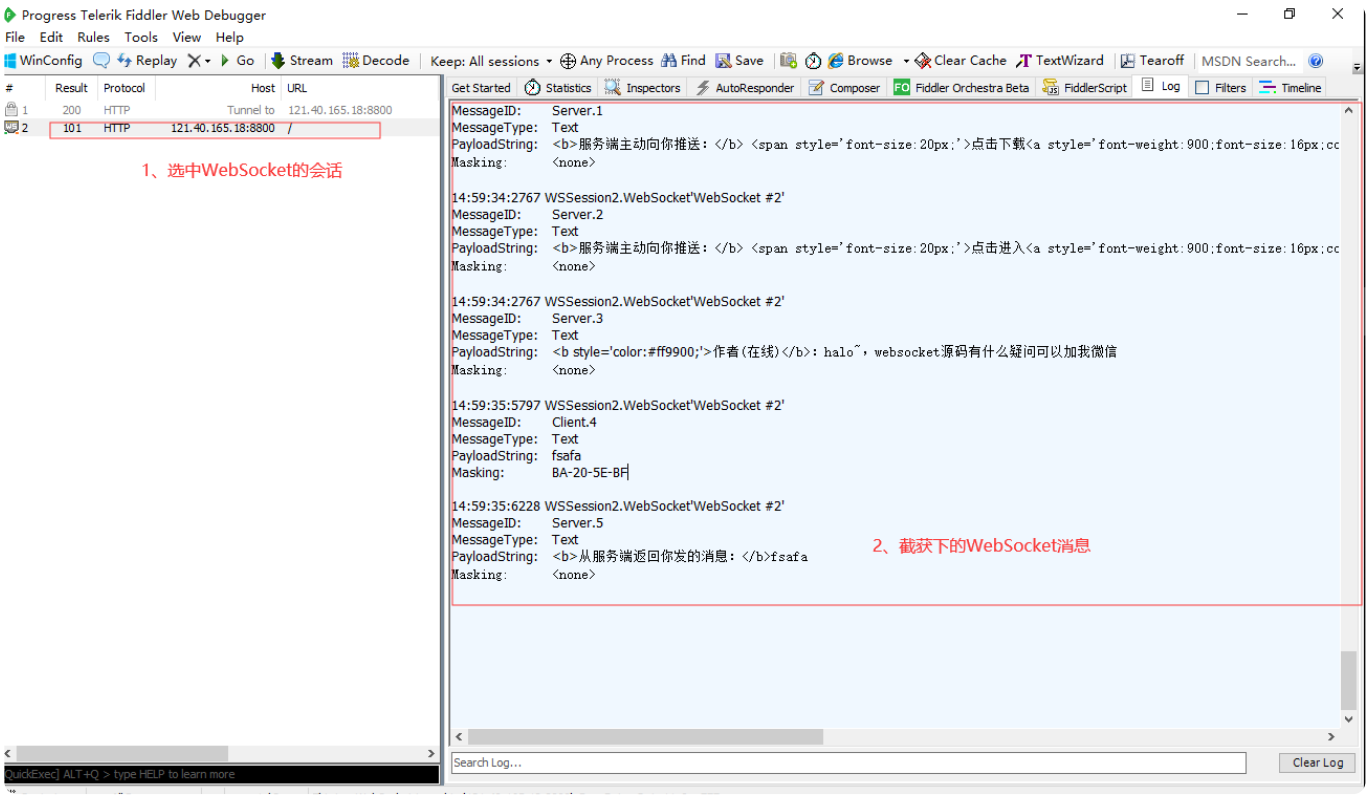
我先做的就是和开发要了他们调用方的代码，当我第一次看到这个代码时，还是很难为情的，因为它是用 Node.js 编写的，当时我对这个技术知之甚少。但凭着自己的经验积累，我多多少少还能看懂一点这个代码，然而我在读了代码之后，发现自己基于这个代码写测试用例并不容易，因为我对 Node.js 技术实在太陌生了，陌生到我无法利用它来完成接口测试。

这种情况我相信你肯定也遇见过，那就是开发工程师很 Nice 地把代码给了你，但你却没办法利用它。但这里我想告诉你的是，面对一个陌生协议的接口测试任务时，无论如何，第一次你还是需要先拿到并了解开发工程师写的客户端代码，因为这样，你就可以对调用方式、参数等接口相关的一些内容有初步印象。在读完相关代码后，你就算是和这些接口完成了首次“会面”，下面你就要想办法敲开接口的大门，让自己能访问被测接口。

由于技术栈问题，我没办法借助开发工程师的力量完成接口测试任务，因此我接下来想到的是，借助一些自己已经熟悉的工具来完成本次测试。我第一个想到的就是我们在之前课程中一起使用过的 Fiddler，因为在任何一个接口项目开始时，无论开发是不是给了我接口文档，我都会先用 Fiddler 访问看一下。

那么 WebSocket 用 Fiddler 怎么搞定？我当时搜索了一下，还真是有办法，具体的办法我就不在这里多说了，其实主要就是修改了 Fiddler 中 Rules 下的 Customize Rules，如果你感兴趣可以自己去搜一下。我只是想告诉你，当你面对陌生技术问题的时候，你应该使用你最熟悉的技术去尝试解决问题。

但从下面的图中你可以看到，虽然我找到了 Fiddler 截获 WebSocket 接口的办法，却不难发现，所截获的全部消息都在日志里面，根本无法操作，所以我想用 Fiddler 完成 WebSocket 测试的想法也就胎死腹中了。



但是，我可以借助 Fiddler 分析 WebSocket 的接口，这也和我们一开始给 Fiddler 这款工具的定位一样，那就是通过它辅助分析我们的被测接口。

自己写 WebSocket 测试代码

当用已有工具基础解决 WebSocket 接口测试这个想法破灭了后，我开始寻求通过编写代码，解决 WebSocket 的接口测试。在这里，我还是建议你要以你自己的技术栈为出发点，寻找解决问题的方法。由于我的主要编程语言是 Python，因此下面一些讲解的示例代码段，我还是以 Python 为例，但是你要知道，解决问题的思路并不限于 Python 的编程语言，它可以是你使用的任何其它语言。

我发现 Python 提供了 WebSocket 的协议库，因此我只要用它完成客户端的撰写，就可以进行接口测试了。这里，我写下了第一个 WebSocket 的调用代码（这里我们以 http://www.websocket.org/demos/echo/ 为例），如下面图中所示，我在代码里面写了详细的注释，你肯定能看懂每一句话的意思。

复制代码

```
#引入websocket的create_connection类
from websocket import create_connection
# 建立和WebSocket接口的链接
ws = create_connection("ws://echo.websocket.org")
# 打印日子
print("发送 'Hello, World'...")
```

```
# 发送Hello, World
ws.send("Hello, World")
# 将WebSocket的返回值存储result变量
result = ws.recv()
# 打印返回的result
print("返回"+result)
# 关闭WebSocket链接
ws.close()
```

不知道你发现没有，上面的代码和 HTTP 协议的接口类似，都是先和一个请求建立连接，然后发送信息。它们的区别是，WebSocket 是一个长连接，因此需要人为的建立连接，然后再关闭链接，而 HTTP 却并不需要进行这一操作。

我相信你肯定还记得在测试框架那一节（04）中，我们学习了一些线性的接口测试代码，然后通过分析这些代码抽象出 Common 类，随着 Common 类的不断丰富，就形成了你自己私有化的测试框架，那么现在问题来了：Common 类中可以也放入 WebSocket 的通用方法吗？

将 WebSocket 接口封装进你的框架

看见上面的代码，我们的第一反应应该是，这里有什么东西可以放到我们自己的 Common 类中呢？你可以按照“测试代码即框架”这一思路将这个 WebSocket 接口封装进你的框架。

我们在前面课程中封装了 Common 类，你可以在它的构造函数中，添加一个 API 类型的参数，以便于知道自己要做的是什么协议的接口，其中 http 代表 HTTP 协议接口，ws 代表 WebSocket 协议接口。由于 WebSocket 是一个长连接，我们在 Common 类析构函数中添加了关闭 ws 链接的代码，以释放 WebSocket 长连接。依据前面的交互流程，实现代码如下所示：

复制代码

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# python代码中引入requests库，引入后才可以在你的代码中使用对应的类以及成员函数
import requests
from websocket import create_connection


# 定义一个common的类，它的父类是object

```

```
class Common(object):
  # common的构造函数
  def __init__(self,url_root,api_type):
    '''
    :param api_type:接口类似当前支持http，ws，http就是http协议，ws是Websocket协议
    :param url_root: 被测系统的跟路由
    '''
    if api_type=='ws':
      self.ws = create_connection(url_root)
    elif api_type=='http':
      self.ws='null'
      self.url_root = url_root


  # ws协议的消息发送

  def send(self,params):
    '''
    :param params: websocket接口的参数

    :return: 访问接口的返回值
    '''
    self.ws.send(params)
    res = self.ws.recv()
    return res


  # common类的析构函数，清理没有用的资源

  def __del__(self):
    '''
    :return:
    '''
    if self.ws!='null":
       self.ws.close()
  def get(self, uri, params=None):
    '''
    封装你自己的get请求，uri是访问路由，params是get请求的参数，如果没有默认为空
    :param uri: 访问路由
    :param params: 传递参数，string类型，默认为None
    :return: 此次访问的response
    '''
    # 拼凑访问地址
    if params is not None:
      url = self.url_root + uri + params
    else:
      url = self.url_root + uri
    # 通过get请求访问对应地址
    res = requests.get(url)
    # 返回request的Response结果，类型为requests的Response类型
    return res
  def post(self, uri, params=None):
```

```
    '''
    封装你自己的post方法，uri是访问路由，params是post请求需要传递的参数，如果没有参数这里为
    :param uri: 访问路由
    :param params: 传递参数，string类型，默认为None
    :return: 此次访问的response
    '''
    # 拼凑访问地址
    url = self.url_root + uri
    if params is not None:
        # 如果有参数，那么通过post方式访问对应的url，并将参数赋值给requests.post默认参数d
      # 返回request的Response结果，类型为requests的Response类型
      res = requests.post(url, data=params)
    else:
      # 如果无参数，访问方式如下
      # 返回request的Response结果，类型为requests的Response类型
      res = requests.post(url)
    return res


def put(self,uri,params=None):
    '''
    封装你自己的put方法，uri是访问路由，params是put请求需要传递的参数，如果没有参数这里为空
    :param uri: 访问路由
    :param params: 传递参数，string类型，默认为None
    :return: 此次访问的response
    '''
    url = self.url_root+uri
    if params is not None:
      # 如果有参数，那么通过put方式访问对应的url，并将参数赋值给requests.put默认参数data
      # 返回request的Response结果，类型为requests的Response类型
      res = requests.put(url, data=params)
    else:
      # 如果无参数，访问方式如下
      # 返回request的Response结果，类型为requests的Response类型
      res = requests.put(url)
    return res


def delete(self,uri,params=None):
    '''
    封装你自己的delete方法，uri是访问路由，params是delete请求需要传递的参数，如果没有参数
    :param uri: 访问路由
    :param params: 传递参数，string类型，默认为None
    :return: 此次访问的response
    '''
    url = self.url_root + uri
    if params is not None:
      # 如果有参数，那么通过put方式访问对应的url，并将参数赋值给requests.put默认参数data
      # 返回request的Response结果，类型为requests的Response类型
      res = requests.delete(url, data=params)
    else:
      # 如果无参数，访问方式如下
```

```
        # 返回request的Response结果，类型为requests的Response类型
        res = requests.put(url)
    return res
```

上面的代码很长，但我的注释很详细，我并不建议你一字不落地都看完，你只要在使用的时候看一下对应的方法就好了。它是一个超级工具集合，最后会变成你自己的类似哆啦 A 梦的万能口袋，你只要做好自己的积累就可以了。

那么，使用上述的 Common 类将上面那个流水账一样的脚本进行改造后，就得出了下面这段代码：

复制代码

```
from common import Common
# 建立和WebSocket接口的链接
con = Common('ws://echo.websocket.org','ws')
# 获取返回结果
result = con.send('Hello, World...')
#打印日志
print(result)
#释放WebSocket的长连接
del con
```

现在，从改造后的代码中，你是不是更能体会到框架的魅力了？它能让代码变得更加简洁和易读，将 WebSocket 的协议封装到你的框架后，你就拥有了一个既包含 HTTP 协议又包含 WebSocket 协议的接口测试框架了，随着你不断地积累新协议，你的框架会越来越强大，你自己的秘密武器库也会不断扩充，随着你对它的不断完善，它会让你的接口测试工作越来越简单，越来越快速。

总结

美好的时光过得都很快，又到了本节课结束的时候了，我今天主要讲了面对一个陌生协议时（比如说 WebSocket），你该如何从零开始完成接口测试任务。

针对一个陌生协议的第一次接口测试，你要保持自己敏锐的测试嗅觉，依据自己的技术基础，尽快解决问题。总地来说，你可以通过三步快速完成测试任务：

1. 借力开发工程师。你首先该借力就是开发工程师，但你不要进入开发工程师给你的那种，从技术基础和理论开始学起，再逐步应用的学习脉络。你要一击致命，直接把他的客户端代码拿来，尽最大可能挪为己用，将其变成自己的接口测试代码。
2. 站在自己的技术栈之上，完成技术积累。如果开发工程师的代码并不能拿来使用，那么你就需要站在自己的技术栈上寻求解决方法，这其中既包含了你已经熟悉的测试工具、测试平台，也包含了自己的测试框架和编码基础。
3. 归入框架。无论你使用哪一种方法，在完成测试工作后，你还是要掌握对应的理论基础，同时想办法将这个一开始陌生的接口，通过自己熟悉的方式合并到你自己的框架中，不断扩充自己框架的测试能力，不断丰富你自己的测试手段。

思考题

我们今天一起学习了如何破解陌生协议接口测试难题的步骤，那么面对 WebSocket 的接口测试任务，结合你现有的技术栈，你是不是也有你自己的解决方案呢？你工作中如果有类似的陌生协议（既可以是第一次接触的协议，也可以是企业私有协议），你是如何解决的呢？欢迎你在留言区中留下你的疑问和你的做法。

我是陈磊，欢迎你在留言区留言分享你的观点，如果这篇文章让你有新的启发，也欢迎你把文章分享给你的朋友，我们一起沟通探讨。

课程学习计划

关注极客时间服务号
每日学习签到

月领 25+ 极客币

【点击】保存图片，打开【微信】扫码>>>

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇　06 | 接口测试平台：工具和框架不可以兼容?

下一篇　08 | 测试数据：是不是可以把所有的参数都保存到Excel中?

精选留言 (9)

写留言

京都念慈菴
2020-02-14

年前刚好公司年会，然后开发了个聊天小程序，就给我测。当时完全不知道怎么搞，就去看这个协议。然后就想先抓包，结果抓不到，捣鼓了修改fiddler抓到了包，然后也是看抓到的数据好像跟平时不一样，然后各种搜索，平时使用的jmeter，就在jmeter中按往常的来试试，正常取样器不行，最后看到jmeter中有个websocket取样器，开心完成压测。

展开

作者回复: 谢谢您的留言，一个解决问题的完美过程。加油

3　3

罗春南_Nancy
2020-02-19

self.ws='null'--加个定义好点
if api_type=='ws':
self.ws = create_connection(url_root)
elif api_type=='http':
self.url_root = url_root...

展开

作者回复: 完美 谢谢您的建议。

1

happychap
2020-02-15

我的尝试的一般顺序：现成工具，工具现成插件，测试框架+项目组封装或采用的api库（如：junit +对应的协议开源库）。
个人觉得遇到这种情况，要会点编程，孰练测试框架胜算才大⊙ᴗ⊙

工作中有突然遭遇xmpp和mqtt协议性能测试的经历，最终jmeter+开源插件+厂商client jar包+demo+api手册搞定了任务。

展开

作者回复: 谢谢您的支持，xmpp和mqtt协议确实不容易遇见，看到您的解决思路可真棒，借力打力得方法会让我们事半功倍。谢谢您的留言，解决方法特别棒，也帮我提供了思考。

1

aoe
2020-02-18

老师的故事很有趣

展开

作者回复: 谢谢

LH Ruby
2020-02-17

ws消息通过jmeter请求发送成功了，但是聊天窗口没有收到消息，请问会是什么原因，现在遇到这个问题，求解中

展开

作者回复: 您好，您的信息量稍微有点少，如果在这里不方便聊天可以来我的blog：https://blog.csdn.net/crisschan留言，因为我有可能需要更多你关于工具的信息。

VeryYoung
2020-02-15

好巧，websocket协议也是我遇到的第一个陌生协议，那个时候测试工期短，用的是java技术栈，就用了netty框架来封装，后面有时间了就进行回顾，发现还有有个websocket.jar开源包，后面又相继遇到了amqp、mqtt等非http协议！

展开

作者回复: 好巧啊，你遇见了很多陌生协议。很宝贵的经验啊

2

shadow

2020-02-15

这里Common里增加了一个入参api_type『def __init__(self,url_root,api_type)』，用于标记请求协议类型，这实际相当于框架底层方法重构，一个方法是给这个入参加默认传参，就是之前的http，不知道还有没有别的方法？
还有一个就是，框架开发，会多多少少的涉及到底层重构，然后可能影响的就是所有的用例都需要修改，这种情况，不知道有没有什么好的规避方法？

展开

作者回复: 您说的实在框架迭代过程中必须经历的，因此为了能够兼容老框架技术栈，我建议在构造函数中给两个参数默认值，默认值就是最开始的uri和api_type的类型，已解决前后兼容的问题。谢谢您这个好问题

2020-02-14

像微服务接口，一般都是使用特殊协议如dubbo、protobuf进行通信的，这种情况应该怎么做测试呢？只能用自己擅长的编程语言写一个客户端模拟调用来进行测试？这种还需要连接配置中心什么的。而且会不会存在有些协议是不跨语言的，比如仅限Java语言，不支持python。

展开

作者回复: 您好，无论您说的dubbo、protobuf还是springcloud，gRPC等都不会受到语言的限制而无法调用的，但是也不能断定没有这个情况，那么就只能靠你自己依据自己的技术栈完成协议的封装了，目前您说的dubbo，protobuf使用各种语言都有对应的解决办法，protobuf可以参考https://www.jianshu.com/p/58513863b491；dubbo可以使用https://blog.csdn.net/cyjs1988/article/details/82495136。

1

东

2020-02-14

你好，我想问一下这里做陌生的协议接口测试，测试人员参考的依据是什么文档，我们怎么得知我们测试的接口准确性呢？

展开

作者回复: 陌生协议只是校验码，参数组成方式等不一样。因此，为了快速完成任务最好的就是看开发写的客户端代码。这样是最快的解决问题的办法。